API-232

RS-232 Application Programming Interface for DOS/Windows 3.x

A powerful RS-232 application programming interface library and utilities for PC/AT and compatible systems

Mar. 1996 (5th Edition)

Copyright Notice

Moxa Technologies Co., Ltd.
Tel: +866-2-8665-6373
Fax: +886-2-8665-6372
www.moxa.com.tw

Moxa Tech USA (CA)
Tel: (408)734-2224
Fax: (408)734-4442
support @moxa.com.tw

Moxa Internet [redacted]

Customer's satisfaction is always our number one concern. To ensure customers get the full benefit of our services, Moxa Internet Services (MIS) have been built for technical support, product inquiry, new driver upgrade, etc.

The followings are the services we provide.

E-mail for technical support
address: support@moxa.com.tw

Ftp site for free driver upgrade
address: moxa.com.tw
user ID: ftp
password: your_email_address

24-hr Gopher server for FAQ
address: moxa.com.tw

World Wide Web (WWW) for product info.
address: http://www.moxa.com.tw

About thi[redacted]

This manual contains information on setting up and using API-232 Application Programming Interface (ver 3.x). Contents are outlined below:

The first two chapters provide an introduction to the API-232 and information on software installation and use of the setup program to configure the boards you will be using. Details on how to install and use the drivers is also included.

Chapter 3 gives complete descriptions of the API-232's functions. A quick reference table can be found at the beginning of the chapter to help you to quickly find the function you need.

Chapter 4 has been included to help you if you should have any problems when using the setup program, the DataScope utilities, or when programming.

Table of [redacted]

Chapter 1
Introduction

The RS-232 communications interface is commonly used for personal computers as well as industrial and lab instrumentation. The "API-232" is a software package which uses the standard RS-232 protocols to control devices under DOS or Windows 3.x. Programming is simplifiedßŠonly one program need be written to control any RS-232 device. Troubleshooting tools are also included to ease system development.

The API-232 software has three major parts: first is the complete software device driver for serial I/O processing and control including interrupt handling, UART control, and memory management; next is the group of interface libraries which allows you to use high-level programming languages for serial communications control; last is a complete set of application programs and utilities, such as DataScope, which allows you to troubleshoot your serial communication more effectively.

A wide range of popular programming languages is supported: Microsoft C, Turbo C, Turbo PASCAL, QuickBASIC, Assembly and CLIPPER under DOS, Visual Basic, Visual C and so on under Windows 3.x.

The DataScope and terminal emulation utilities allow you to troubleshoot the serial communication easily. DataScope allows you to monitor or log data between two devices and edit the acquired data from an easy-to-use menu-based user interface. The terminal emulation program allows simulation and manual control of computer or device communication operations. Diagnostic functions ease the process of installation.

Chapter 2 Installation and Setup

The API-232 includes an installation program which lets you copy the drivers and library routines into your hard disk. We recommend making a copy of the original software diskette and keeping the original in a safe place.

2.1 For DOS users

Insert the API-232 software diskette into the floppy drive, change directory (cd) to the drive then type

DOSINST

then an installation screen will appear. Enter the target directory the software driver will be copied to. Press F2 to start the installation.

Next, simply go to the drive or directory containing the program files (C:\MOXA\BIN, for example) and type

SETUP

at the DOS prompt. This program is to configure (IRQ, port number...) the MOXA boards. The configuration will be refered to by the later executed DOS driver. Note that if you are using MOXA new generation ASIC boards such as C102, C104 or C168, then execute IO-IRQ to designate I/O address and IRQ before running SETUP program.

The DRIVER SELECTION screen will then appear. The DRIVER SELECTION screen shows names of the boards which can be setup. If the name of the board is grayed, that means that the board is unavailable.

Use arrow key to highlight the board you want to set up and hit Page Down or Enter to select it. An online instructive help is always available at the bottom of each sub-window.

2.1.1 Setting up C102/C104/C168

MOXA supports device driver for 2/4/8 serial ports boards such as C102, C104, C168 family and other manufacturer's non-intelligent multiport boards that use 16450 or 16550 UART. The maximum supported number of serial ports is limited to 8. For example, you can install one C168 (1ßď8 ports) or two C104s (2ßď4 ports) or four C102s (4ßď2 ports) under the DOS system.

Choose "C102/C104/C168 Series Multiport Board" to do the setup. On this screen, you will have to enter or modify each port's configuration. These are the port initial value when driver is loaded. Press F3: "Add port" to start the process. Some noticeable fields are explained below.

Port number: This is actually the port ID of each port. The application software will have to refer to the port by its port number (ID). Duplicated port number is not allowed.

Base I/O address: The I/O address of each port. Overlapped or duplicated I/O address is not allowed.

Interrupt number: The IRQ number of each port. Several ports may share one common IRQ.

TxD buffer size: The transmit (output) buffer reserved for the port.

RxD buffer size: The receive (input) buffer reserved for the port.

An instructive help at the bottom of the sub-window is used to ease the configuration. Press F1: "Help" for more information.

Driver Loading

Having completed the setup, load the TSR driver at the DOS prompt (or from within \AUTOEXEC.BAT batch file),

SER-DRV

To support various types of UARTs such as 16450, 16550 and 16550C, several options for SER-DRV are provided. Type SER-DRV/? to see the available options.

The driver will detect the multiport board automatically. If the board is detected, a message similar to below will show:

Universal 2/4/8 Serial Port Communication Driver (Ver x.xx)
Device Driver Setup O.K.

Which means the board and the driver have been successfully installed. At this point, you are ready to execute application that supports API-232 functions, or start developing your own application.

If there is no matched port, the screen will show a message like the one below:

Universal 2/4/8 Serial Port Communication Driver (Ver x.xx)
No Serial Port Found!!

In this case, refer to "Troubleshooting" chapter for possible reasons and solutions.

Driver Removal

To remove the C102/C104/C168 driver from memory, type the following at the DOS prompt,

SER-DRV/Q

2.1.2 Setting up C218/C320

MOXA supports device driver for intelligent multiport boards: C218 and C320. The maximum supported number of C218 plus C320 boards is limited to 4 in one system. For example, you can install one C218 and three C320s, or two C218s and two C320s, or four C320s under the DOS system. Note that the C218/C320 can coexist with C102, C104 or C168 series in the same system.

Choose "C218/C320 Async Multiport Board" to do the setup.

Type: Choose the board type, C218 or C320

Modules: The number of 8-port UART Modules of C320. The Modules is 1 for C218.

Port number: This is actually the port ID of each port. The application software will refer to the port by its port number (ID). Duplicated port number is not allowed.

Memory base address: The base address of each board. Duplication is not allowed.

Interrupt number: The IRQ number of the board. All C218s and C320s must share one IRQ only.

On the second screen titled as "PORT SETUP", enter/modify each ports configu-ration. These are the port initial value when driver is loaded. Some noticeable fields are explained below.

On board TxD buf: The transmit (output) buffer already allocated on the C218/C320 board for each port. Cannot be altered.

On board RxD buf: The receive (input) buffer already allocated on the C218/C320 board for each port. Cannot be altered.

External RxD buf: Extra receive (input) buffer you wish to allocat for each port. This will consume the DOS conventional memory. Default value is 0.

Driver Loading

Having completed the setup, load the TSR driver at the DOS prompt (or from within \AUTOEXEC.BAT batch file),

MX-DRV

The driver will detect the multiport board automatically. If the board is detected, a message similar to below will show:

MOXA C218/C320 Communication Driver (Ver x.xx)
Setup O.K.

Which means the board and the driver is installed properly. At this point, you are ready to execute application that supports API-232 function calls, or start developing your own application.

If the board does not match the setup data, the screen will show a message like the one below,

MOXA C218/C320 Communication Driver (Ver x.xx)
MOXA Board Not Found!!

In this case, refer to "Troubleshooting" chapter for possible reasons and solutions.

Driver Removal

To remove the C218/C320 driver from memory, type the following at the DOS prompt,

MX-DRV/Q

2.2 For Windows 3.x users

This section describes only the software driver for Windows 3.x. If you are a Windows NT or Windows 95 user, please contact MOXA dealer/distributor for how to obtain the software drivers.

There are two kinds of Windows drivers supported by MOXA. One is the Standard Windows COMM Driver; the other is the MOXA Proprietary Driver. Choose one of the two drivers for your development platform.

The following is the matrix of Windows drivers and supported MOXA boards.

	Standard COMM Driver	**Proprietary Driver**
C218	✓	✓
C320	✓	✓
C104	✓	
Other 4 port boards	✓	

The Standard Windows COMM Driver supports Microsoft Windows COMM API (Application Programming Interface) such as OpenComm(), ReadComm() and WriteComm(). Application software like Windows Terminal program, pcANYWHERE for Windows or other programs that support Windows COMM API calls can communicate to outside world via the MOXA multiport boards. Up to nine serial ports are supported due to the limitations of Windows COMM API.

The MOXA Proprietary Driver is actually a Dynamic Linking Library (DLL). It supports MOXA API-232 functions such as sio_open(), sio_read() and sio_write(). This is quite convenient for those users who wish to uqgrade their existing DOS API-232 applications to Windows environment. In addition, the Proprietary Driver does not restrict to nine serial ports, instead it can support up to 128 ports (4 C320 boards in this case).

Carefully examine which one of the two drivers is suitable for you. Here is our suggestion: If you need no more than 9 serial ports, or intend to run a ready-made software program (such as pcANYWHERE for Windows) that supports Windows COMM API calls, then choose Standard Windows COMM driver as your platform.

If you need more than 9 serial ports or have been familiar with the MOXA API-232 functions, then MOXA Proprietary Driver would be suitable for you. In this case, developing the Windows application by yourself is necessary.

2.2.1 Installing the Standard Windows COMM Driver

Insert the API-232 software diskette into the floppy drive A: (or B:). At Windows File Manager, on A: (or B:) execute

WININST

A "Driver Installation" sub-window will appear. Choose board type, driver type and the working directory where the software will be copied to.

MOXA supports standard Windows COMM drivers for C320, C218, C104 and other 4 port non-intelligent boards. Due to the limitations of the Microsoft Windows COMM API, the driverr will support a maximum of 9 ports, COM1-COM9. Note that, when using C104 or other 4 port non-intelligent boards, maximum 6 ports is supported if the existing standard COM ports (COM1 and COM2) are included.

Examples: One serial port on the motherboard set for COM1 (0x3F8, IRQ4), and 8 ports, COM2-COM9, on a C218 board set for IRQ11;

One serial port, on the motherboard set for COM1 (0x3F8, IRQ4), 4 ports, COM2-COM5, on a C104 board set for IRQ3.

A serial mouse, if used, must be installed on either COM1 (0x3F8, IRQ4) or COM2 (0x2F8, IRQ3), and must have its own dedicated IRQ.

After completing the installation, restart the Windows system. You will find an entry "comm.drv = sercomm.drv" for C104 or "comm.drv = mxcomm.drv" for C320/C218 in Windows SYSTEM.INI file, [boot] section. Also, there will be a Windows group "MOXA Standard COMM Driver" generated for reconfiguration, driver removal and so on.

At this point, you are ready to execute applications that support Windows COMM API calls. Note that the MOXA DOS device driver MX-DRV or SER-DRV must not be loaded concurrently with the Standard Windows COMM Driver.

2.2.2 Installing the MOXA Proprietary Driver

Insert the API-232 software diskette into the floppy drive A: (or B:). At Windows File Manager, on A: (or B:) execute

WININST

A "Driver Installation" sub-window will appear. Choose "C320/C218 series" as the board type and "MX Proprietary Driver (API-232)" as the driver type since only C320/C218 support the Proprieyary Driver (API-232).
Follow the instruction or on-line help to install and configure the driver. After completing the installation, there will be a Windows group "MX Proprietary Driver" generated for reconfiguration, driver removal and so on.

At this point, you are ready to develop the Windows application by calling the API-232 library functions described on Chapter 3. (Restarting the Windows system is not necessary) Note that the MOXA DOS driver MX-DRV must not be loaded concurrently with the MOXA Proprietary Windows Driver.

Chapter 3
Library Functions

This chapter lists all the API-232 function calls under DOS and MS Windows 3.x. The detail function, syntax and example of each library are also given.

To provide software developer a more comprehensive and easier development tool, some functions are modified and many useful functions are added from API-232 Version 3.0. For example, functions for modem control and file transfer protocols (ZMODEM, YMODEM, etc.) are added.

The API-232 3.x is compatible with the API-232 2.x at the application binary code level except sio_getbaud() function. That means, executable application program developed under API-232 2.x can be directly ported to API-232 3.x environment without any modification, compiling or linking provided that the sio_getbaud() function is not used. For those who upgrade from API-232 1.x, linking again the application object with API-232 3.x library is necessary.

3.1 Quick Reference Table

The function list is organized in such a way that the similar functions are grouped together. There are nine catagories: Driver Manipulation, Port Control, Input Data, Output Data, Port Status Inquiry, Interrupt Control, Modem Control, File Transfer, Miscellaneous Functions.

Note: (M) behind a function index means function Modified;
(N) behind a function index means New function in comparision to API-232 2.x, 1.x.

Driver Manipulation

Index	Function	Description
1	sio_init()	Initialize Windows DLL driver (only for Windows 3.x)
2	sio_end()	Release Windows DLL driver (only for Windows 3.x)
3	sio_reset()	Reset all ports

Port Control

4	sio_open()	Start port to receive/transmit data
5	sio_close()	Stop port receiving/transmiting data
6	sio_ioct1()	Set port baud rate, parity, etc.
7	sio_flowctrl()	Set port H/W and/or S/W flow control
8	sio_flush()	Flush input and/or output buffer
9 (N)	sio_DTR()	Set DTR state
10 (N)	sio_RTS()	Set RTS state
11	sio_lctrl()	Set both DTR and RTS state
12 (N)	sio_baud()	Set baud rate using the actual speed value
13	sio_disableTx()	Stop port from transmiting data
14	sio_enableTx()	Enable port to transmit data again
15 (N)	sio_disableRx()	Stop port from receiving data
16 (N)	sio_enableRx()	Enable port to receive data again
17	sio_overlap()	Set driver's input buffer overlap mode when buffer's overflow
18(N)	sio_ignore_errdata()	Discard data if received error

Input Data

19	sio_getch()	Read one character at a time from driver's input buffer
20	sio_read()	Read a block of data from driver's input buffer
21	sio_linput()	Read a block of data that ends with a terminator character
22	sio_linput_t()	Same as sio_linput(), add timeout limitation

Output Data

23	sio_putch()	Write one character at a time to driver's output buffer
24	sio_putb()	Write a block of data (either whole block or nothing written)
25	sio_write	Write a block of data (probably only partial block written)
26	sio_putb_t()	Same as sio_putb(), add timeout limitation
27(N)	sio_putb_x()	Write a block of data (especially for RS-485 communication)

Port Status Inquiry

28	sio_lstatus()	Get line status
29	sio_iqueue()	Size of data accumulated in driver's input buffer
30	sio_oqueue()	Size of data not yet sent out (still kept in driver's output buffer)
31	sio_ifree()	Size of free space in driver's input buffer
32	sio_ofree()	Size of free space in driver's output buffer
33	sio_Tx_hold()	Check the reason why data could not be transmitted
34 (M)	sio_getbaud()	Get the baud rate setting
35	sio_getmode()	Get the parity, data bits... settings

36	sio_getflow()	Get the H/W ans S/W flow control settings
37	sio_brk_cnt()	Get the counter value of received BREAK and then clear
38	sio_overflow()	Check if driver's input buffer overflow
39 (N)	sio_data_status()	Check if any error happen when receiving data

Interrupt Control

40	sio_term_irq()	Set interrupt service routine when terminator character received
41	sio_cnt_irq()	Set interrupt service routine when certain amount of data received
42	sio_modem_irq()	Set interrupt service routine when line status changed
43	sio_break_irq()	Set interrupt service routine when break signal received

Modem Control

The API-232 supports C library for Hayes compatible modem control. Please include "MODEM-C.H" header file in your source code. Link one of the following libraries to your application object.

MODEM-CS.OBJ (small model)
MODEM-CM.OBJ (medium model)
MODEM-CC.OBJ (compact model)
MODEM-CL.OBJ (large model)

44 (N)	sio_McAnswer()	Answer the caller manually
45 (N)	sio_McAutoAnswer()	Set auto answer after a number of rings

46 (N)	sio_McCommandMode()	Tell the modem to go off line
47 (N)	sio_McDataMode()	Tell the modem to go on line
48 (N)	sio_McDial()	Dial a phone number and wait for carrier
49 (N)	sio_McGetSRegister()	Read the value of the modem's S register
50 (N)	sio_McHangUp()	Hang up the line
51 (N)	sio_McIsConnect()	Check if the line is connected
52 (N)	sio_McOffHook()	Tell the modem to go off hook
53 (N)	sio_McReset()	Performs a reset of the modem
54 (N)	sio_McRings()	Read how many rings has come in
55 (N)	sio_McSetSRegister()	Send a register set command to the modem
56 (N)	sio_WaitConnect()	Check if the line is connected, if not, wait for timeout

File Transfer

The API-232 only supports C library for file transfer. Please include "FILE-C.H" header file in your source code. Link one of the following libraries to your application object.

FILE-CS.OBJ (small model)
FILE-CM.OBJ (medium model)
FILE-CC.OBJ (compact model)
FILE-CL.OBJ (large model)

57 (N)	sio_FtASCIIRx()	Receive a file using ASCII protocol
58 (N)	sio_FtASCIITx()	Transmit a file using ASCII

		protocol
59 (N)	sio_FtKermitRx()	Receive files using KERMIT protocol
60 (N)	sio_FtKermitTx()	Transmit files using KERMIT protocol
61 (N)	sio_FtXmodem1KCRCRx()	Receive a file using XMODEM, 1K CRC
62 (N)	sio_FtXmodem1KCRCTx()	Transmit a file using XMODEM, 1K CRC
63 (N)	sio_FtXmodemCheckSumRx()	Receive a file using XMODEM, checksum
64 (N)	sio_FtXmodemCheckSumTx()	Transmit a file using XMODEM, checksum
65 (N)	sio_FtXmodemCRCRx()	Receive a file using XMODEM, CRC
66 (N)	sio_FtXmodemCRCTx()	Transmit a file using XMODEM, CRC
67 (N)	sio_FtYmodemRx()	Receive files using YMODEM protocol
68 (N)	sio_FtYmodemTx()	Transmit a file using YMODEM
69 (N)	sio_FtZmodemRx()	Receive files using ZMODEM protocol
70 (N)	sio_FtZmodemTx()	Transmit a file using ZMODEM

Miscellaneous Functions

71	sio_getports()	Get no. of ports recognized by the driver
72	sio_break()	Send out BREAK signal
73	sio_bank()	Get the MOXA board memory mapping address
74	sio_timeout()	Set timeout value for sio_linput_t() and sio_putb_t()
75	sio_loopback()	Port loopback selftest

Cross Reference

Function Name	Function Index

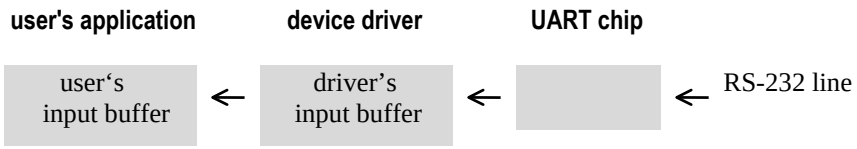
3.2 Buffer Concept

There are two kinds of buffer need to be understood in writing application software that

calls API-232 functions. One is the buffer manipulated by the device driver, or "driver's buffer". The other is the buffer managed by user's application, or "user's buffer".

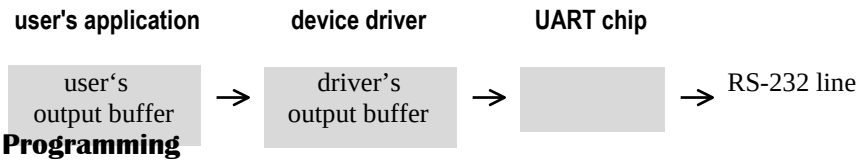
Input

After loading the device driver, the driver starts to read the character queued in the UART chip if the port is properly initialized. It then puts the character into the "driver's input buffer" which is maintained by the driver itself. The data are kept in "driver's input buffer" until the user's application calls API-232 function (e.g. sio_read()) to read to the "user's input buffer".



Output

To output data, the user's application first prepare the data in the "user's output buffer". It then put the data into the "driver's output buffer" by calling API-232 function (e.g. sio_putb()). The driver will continue to send out data in the "driver's output buffer" to UART chip character by character.



3.3 Programming

A simplest program that calls API-232 functions would have a similar structure as follows.

- Calling Driver Manipulation functions
- Calling Port Control functions

- Calling Input/Output Data functions
- Calling Port Status functions if necessary

For example, following illustrates a piece code of a simple API-232 calls in C language. (Variables are not defined)

```
/* i = sio_init();    */      /* If under Windows 3.x, uncomment this func.*/
i = sio_reset();               /* Driver Manipulation function                */
if (i <= 0) {
        printf ("No driver found!\n");
        exit();
}
i = sio_open(port);                    /* Port Control, enable the port       */
if (i == 0) {
        printf ("port opened ok.\n");
        sio_ioctl(port, baud, mode);   /* Port Control,set baud,parity...     */
        sio_putb(port, "ABCDE", 5); /* Output Data function                   */
        sio_read(port, ibuf, length);  /* Input Data function                 */
}
```

3.4 Function Descriptions

Function 1 sio_init()

Driver initialization for MS Windows 3.x. The DLL driver is invoked, the board's configuration file is read, and then the driver will find the multiport board and do the port initialization.

Language	Command
C	---
PASCAL	---
QBASIC	---
CLIPPER	---
Assembly	---
Windows	int FAR PASCAL sio_init()

Argument:

None

Return:

-1	user not running in 386 Enhanced mode
-2	MX-WIN.CNF file format error
-3	not enough space for extended rcv buffer
-4	extended rcv buffer can't be locked
-5	can't find free memory bank
0	no board found
1-4	number of boards found

Note: If there is an error or no board is found, further functions will not work.

Example: (Windows C)

```
int i;
i = sio_init();
```

Function 2 sio_end()

Release the MOXA Windows 3.x driver and frees all allocated memory.

Language	Command
C	---
PASCAL	---
QBASIC	---

CLIPPER	---
Assembly	---
Windows	void FAR PASCAL sio_end()

Return:

None

Function 3 sio_reset()

Reset all serial ports, disables all interrupt service routines, and flushes the driver's input/output buffers.

Language	Command
C	int sio_reset()
PASCAL	sio_reset:integer
QBASIC	SioReset(drvno%)

CLIPPER	sio_reset()
Assembly	_ _sio_reset
Windows	int FAR PASCAL sio_reset()

Argument:

None

Return:

integer n (ax)
n > 0: total number of drivers
n <=0: No driver found

Example: (C language)

```
int stat;
if ((stat = sio_reset()) <= 0)
        printf ("No Driver Found!\n");
```

Function 4 sio_open()

Enable a serial port for transmit/receive. The port must be enabled by issuing this function before the following operations: sio_getch, sio_linput, sio_read, sio_iqueue, sio_ifree, sio_putch, sio_putb, sio_write, sio_oqueue, sio_ofree, sio_flush, sio_disableTx, sio_enableTx, sio_putb_t, sio_linput_t, sio_term_irq, sio_cnt_irq, sio_modem_irq, sio_break_irq, sio_putb_x.

Language	Command
C	sio_open(int port)

PASCAL	sio_open(port:integer) :integer
QBASIC	SioOpen(port%, er%)
CLIPPER	sio_open(port)
Assembly	_ _sio_open
Windows	int FAR PASCAL sio_open(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
stat = sio_open(port);
if (stat == 0)
        printf ("Port 1 opened\n");
```

Function 5 sio_close()

Disable a serial port so that it cannot receive/transmit data.

Language	Command
C	sio_close(int port)
PASCAL	sio_close(port:integer) :integer
QBASIC	SioClose(port%, er%)
CLIPPER	sio_close(port)
Assembly	_ _sio_close
Windows	int FAR PASCAL sio_close(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n >= 0: ok
0ßĎno data in Tx/Rx buffer when the port is closed
1ßĎsome data still in Rx buffer
2ßĎsome data still in Tx buffer
3ßĎsome data still in Tx and Rx buffer
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
stat = sio_close(port);
if (stat == 0)
        printf ("Port 1 closed\n");
```

Function 6 sio_ioctl()

Control the setting of the serial port's I/O control register, such as baud rate, parity, data bits and stop bits.

Language	Command
C	sio_ioctl(int port, int baud, int mode)
PASCAL	sio_ioctl(port, baud, mode:integer) :integer
QBASIC	SioIoctl(port%, baud%, mode%, er%)
CLIPPER	sio_ioctl(port, baud, mode)
Assembly	_ _sio_ioctl
Windows	int FAR PASCAL sio_ioctl(port, baud, mode)

Argument:

int port, baud, mode;

port (dx) = port number

baud (ch) = 0 = 50	6 = 600	12 = 9600
1 = 75	7 = 1200	13 = 19200
2 = 110	8 = 1800	14 = 38400
3 = 134.5	9 = 2400	15 = 57600
4 = 150	10 = 4800	
5 = 300	11 = 7200 (bits/sec)	

Note: Some MOXA boards may not support 57600 bps. Please refer to the board's manual for information.

mode (cl) = bit_cnt OR stop_bit OR parity

bit_cnt (bit 0, 1) = 0x00 = bit_5
0x01 = bit_6
0x02 = bit_7
0x03 = bit_8

stop_bit (bit 2) = 0x00 = stop_1
0x04 = stop_2

parity (bit 3, 4, 5) = 0x00 = none
0x08 = odd
0x18 = even
0x28 = mark
0x38 = space

Return:

int n (ax) = 0: ok

n = -1: no such port
n = -2: can't control MOXA board
n = -3: bad function call

Example: (C language)

```
#include "head-c.h"
int port = 0;      /* set up 2400 baud, none parity, 8 data bit, 1 stop bit    */
if (sio_ioctl( port, B2400, P_NONE | BIT_8 | STOP_1) == 0)
        printf ("Setting port #%d:B2400,n,8,1\n", port);
```

Function 7 sio_flowctrl()

Set hardware and/or software flow control.

Language	Command
C	sio_flowctrl(int port, int mode)
PASCAL	sio_flowctrl(port, mode:integer) :integer
QBASIC	SioFlowCtrl(port%, mode%, er%)
CLIPPER	sio_flow(port, mode)
Assembly	_ _sio_flowctrl
Windows	int FAR PASCAL sio_flowctrl(port, mode)

Argument:

int port, mode;

port (dx) = port number
mode (al)
bit 0: CTS flow control
bit 1: RTS flow control
bit 2: Tx XON/XOFF flow control
bit 3: Rx XON/XOFF flow control
(0 = OFF 1 = ON)

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1, mode = 3;    /* set H/W flow control */
stat = sio_flowctrl(port, mode);
```

Function 8 sio_flush()

Flush the driver's input/output buffer. The data will no longer exists.

Language	Command
C	sio_flush(int port, int func)
PASCAL	sio_flush(port, func:integer) :integer
QBASIC	SioFlush(port%, func%, er%)
CLIPPER	sio_flush(port, func)
Assembly	_ _sio_flush
Windows	int FAR PASCAL sio_flush(port, func)

Argument:

```
int port, func;

port (dx) = port number
func (ax) = flush function
            0 : flush input buffer
            1 : flush output buffer
            2 : flush input & output buffer
```

Return:

```
int n (ax)

n =  0:   ok
n = -1:   no such port
n = -2:   can't control MOXA board
n = -3:   bad function call
n = -5:   port is not open
```

Example: (C language)

```
int stat, port; /* flush input queue */
stat = sio_flush(port, 0);
```

Function 9 (N) sio_DTR()

Set the DTR state of a port.

Language	Command
C	sio_DTR(int port, int mode)
PASCAL	sio_DTR(port, mode:integer) :integer
QBASIC	SioDTR(port%, mode%, er%)
CLIPPER	sio_DTR(port, mode)
Assembly	_ _sio_DTR
Windows	int FAR PASCAL sio_DTR(port, mode)

Argument:

```
int port, mode;

port (dx) = port number
mode (ax) = 0: turn DTR off
            1: set DTR on
```

Return:

```
int n (ax)

n=  0:   ok
n= -1:   no such port
n= -3:   bad function call
```

Example: (C language)

```
#include "head-c.h";
int stat, por t= 1;
stat = sio_DTR(port, 1);          /* to set DTR on */
```

Function 10 (N) sio_RTS()

Set the RTS state of a port.

Language	Command
C	sio_RTS(int port, int mode)
PASCAL	sio_RTS(port, mode:integer) :integer
QBASIC	SioRTS(port%, mode%, er%)
CLIPPER	sio_RTS(port, mode)
Assembly	_ _sio_RTS
Windows	int FAR PASCAL sio_RTS(port, mode)

Argument:

int port, mode;

port (dx) = port number
mode (ax) = 0: turn RTS off
1: set RTS on

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call
n = -6: can't control because the port is set as auto H/W flow control by sio_flowctrl()

Example: (C language)

```
#include "head-c.h";
int stat, port = 1;
stat = sio_RTS(port, 1);                /* to set RTS on */
```

Function 11 sio_lctrl()

Set both the DTR and RTS state.

Language	Command
C	sio_lctrl(int port, int mode)
PASCAL	sio_lctrl(port, mode:integer) :integer
QBASIC	SioLctrl(port%, mode%, er%)
CLIPPER	sio_lctrl(port, mode)
Assembly	_ _sio_lctrl
Windows	int FAR PASCAL sio_lctrl(port, mode)

Argument:

int port, mode;

port (dx) = port number
mode (ax) = C_DTR (bit 0), C_RTS (bit 1)

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -2: can't control the board
n = -3: bad function call
n = -6: can't control because the port is set as auto H/W flow control by sio_flowctrl()

Example (C language):

```
#include "head-c.h";
int stat, port = 1;
stat = sio_lctrl(port, C_DTR | C_RTS);
```

Function 12 (N) sio_baud()

Set baud rate using the actual speed value.

Language	Command
C	sio_baud(int port, long speed)
PASCAL	sio_baud(port:integer, speed: long) :integer
QBASIC	SioBaud(port%, speed%, er%)
CLIPPER	sio_baud(port, speed)
Assembly	_ _sio_baud
Windows	int FAR PASCAL sio_baud(port, speed)

Argument:

int port; long speed;

port (dx) = port number
speed (ax) = true baud rate, e.g. 200, 1200, 9600, 19200, etc.

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
#include "head-c.h";
int stat, port = 1;
stat = sio_baud(port, 200);     /* to set port speed to 200 bps */
```

Function 13 sio_disableTx()

Stop the port from transmiting data.

Language	Command
C	sio_disableTx(int port)
PASCAL	sio_disableTx(port:integer) :integer
QBASIC	SioDisableTx(port%, er%)
CLIPPER	sio_disTx(port)
Assembly	_ _sio_disableTx
Windows	int FAR PASCAL sio_disableTx(port)

Argument:

```
int port;

port (dx) = port number
```

Return:

```
int n (ax)

n =  0:   ok
n = -1:   no such port
n = -2:   can't control MOXA board
n = -3:   bad function call
n = -5:   port is not open
```

Example: (C language)

```
int stat, port = 1;
stat = sio_disableTx(port);
if (stat == 0)
        printf ("Port 1 transmit disabled\n");
```

Function 14 sio_enableTx()

Enable the port to transmit data again.

Language	Command
C	sio_enableTx(int port)
PASCAL	sio_enableTx(port:integer) :integer
QBASIC	SioEnableTx(port%, er%)
CLIPPER	sio_enTx(port)
Assembly	_ _sio_enableTx
Windows	int FAR PASCAL sio_enableTx(port)

Argument:

```
int port;

port (dx) = port number
```

Return:

```
int n (ax)

n =  0:   ok
n = -1:   no such port
n = -2:   can't control MOXA board
n = -3:   bad function call
n = -5:   port is not open
```

Example (C language):

```
int stat, port = 1;
stat = sio_enableTx(port);
if (stat == 0)
        printf ("Port 1 transmit enabled\n");
```

Function 15 (N) sio_disableRx()

Stop the port from receiving data.

Language	Command
C	sio_disableRx(int port)
PASCAL	sio_disableRx(port:integer) :integer
QBASIC	SioDisableRx(port%, er%)
CLIPPER	sio_disRx(port)
Assembly	_ _sio_disableRx
Windows	int FAR PASCAL sio_disableRx(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -2: can't control MOXA board
n = -3: bad function call
n = -5: port is not open

Example (C language):

```
int stat, port = 1;
stat = sio_disableRx(port);
if (stat == 0)
printf ("Port 1 receival data disabled\n");
```

Function 16 (N) sio_enableRx()

Enable the port to receive data again.

Language	Command
C	sio_enableRx(int port)
PASCAL	sio_enableRx(port:integer) :integer
QBASIC	SioEnableRx(port%, er%)
CLIPPER	sio_enRx(port)
Assembly	_ _sio_enableRx
Windows	int FAR PASCAL sio_enableRx(port)

Argument:

```
int port;

port (dx) = port number
```

Return:

```
int n (ax)

n =  0:   ok
n = -1:   no such port
n = -2:   can't control MOXA board
n = -3:   bad function call
n = -5:   port is not open
```

Example: (C language)

```
int stat, port = 1;
stat = sio_enableRx(port);
if (stat == 0)
        printf ("Port 1 receiving data enabled\n");
```

Function 17 sio_overlap()

Set the data-overlap made for the case when the driver's input buffer is full.

Language	Command
C	sio_overlap(int port, int mode)
PASCAL	sio_overlap(port, mode:integer) :integer
QBASIC	SioOverlap(port%, mode%, er%)
CLIPPER	sio_olap(port, mode)
Assembly	_ _sio_overlap
Windows	int FAR PASCAL sio_overlap(port, mode)

Argument:

int port, mode;

port (dx) = port number
mode = 0: when Rx buffer is full, received data cannot overlap. Last received data will be lost.
mode = 1: when Rx buf full, received data can overlap. First received data in input buffer will be lost.

Return:

int n (ax)

n = 0: ok
n = -1: no such port

Example: (C language)

```
/*    enable buffer overlap function */
int port = 1, mode = 1
stat = sio_overlap(port, mode);
```

Function 18 (N) sio_ignore_errdata()

Discard data if received data error.

Language	Command
C	sio_ignore_errdata(int port, int mode)
PASCAL	sio_ignore_errdata(port, mode:integer) :integer
QBASIC	SioIgnoreErrdata(port%, mode%, er%)
CLIPPER	sio_ignore_errdata(port, mode)
Assembly	_ _sio_ignore_errdata
Windows	int FAR PASCAL sio_ignore_errdata(port, mode)

Argument:

int port, mode;

port (dx) = port number
mode (ax) = 0: accept data even error
1: ignore (or discard) data if error

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example (C language):

```
#include "head-c.h";
int stat, port = 1;
stat = sio_ignore_errdata(port, 1);    /* discard data if error */
```

Function 19 sio_getch()

Read one character from the driver's input buffer.

Language	Command
C	sio_getch(int port)
PASCAL	sio_getch(port:integer) :integer
QBASIC	SioGetch(port%, ch%)
CLIPPER	sio_getch(port)
Assembly	_ _sio_getch
Windows	int FAR PASCAL sio_getch(port)

Argument:

int port (dx) = port number

Return:

int n (ax)

n = -1: no such port
n = -3: bad function call
n = -4: EOL , no data read
n = -5: port is not open
n = 0

Example (C language):

```
int stat, port; if ((stat =
        sio_getch(port)) >= 0) printf ("read character [%c]", stat);
```

Function 20 sio_read()

Reads data from the driver's input buffer. If the length of data in the driver's input buffer is less than the user's buffer, then all data in the driver's input buffer will be transferred to the user's buffer. Otherwise, only 'len' bytes will be transferred to the user's buffer.

Language	Command
C	sio_read(int port, char *buf, int len)
PASCAL	sio_read(port : integer) :string
QBASIC	SioRead(port%, buf$, cnt%)
CLIPPER	sio_read(port, buf)
Assembly	_ _sio_read
Windows	int FAR PASCAL sio_read(port, char far *buf, len)

Argument:

int port, len; char *buf;

port (dx) = port number
buf (es:di) = receive buffer pointer
len (cx) = buffer length

Return:

int n (ax)

n > 0: length of data received
n = 0: no data received
n = -1: no such port
n = -3: bad function call
n = -5: port is not open

Example: (C language)

```
#define LEN 80; char buf [LEN];
int stat, port;
stat = sio_read(port, buf, LEN);
if (stat > 0)
        printf ("%d characters received\n", stat);
```

Function 21 sio_linput()

Read a block of data from the driver's input buffer until the terminator character is encountered or "len" bytes of data are read.

Language	Command
C	sio_linput(int port, char *buf, int len, int term)
PASCAL	sio_linput(port, term:integer) :string
QBASIC	SioLinput(port%, buf$, term%, cnt%)
CLIPPER	sio_linput(port, buf, term)
Assembly	_ _sio_linput
Windows	int FAR PASCAL sio_linput(port, char far *buf, len, term)

Argument:

```
int port, len, term;
char *buf;

port (dx) = port number
buf (es:di) = receive buffer pointer
len (cx) = buffer length
term (al) = terminator character
```

Return:

```
int n (ax)

n = -1:  no such port                    n = -5:  port is not open
n = -3:  bad function call               n => 0:  length of data received
```

Example: (C language)

```
I = sio_linput(0, buf, 10, 13);

/* Read data from port 0, max length is 10 bytes, terminator is CR (ASCII 13).
If less than 10 characters accumulated in driver's input buffer and a CR is in the
5th byte, this function will read 5 bytes, including CR.  If more than 10 bytes are
ueued in the buffer without a CR, it will read the first 10 bytes into the user buffer
`buf'. */
```

Function 22 sio_linput_t()

Read block of a data from the drivers input buffer until the termination character is encountered or a timeout occurs. (See also the sio_linput() function.)

Language	Command
C	sio_linput_t(int port, char *buf, int len, int term)
PASCAL	sio_linput_t(port, term:integer) :string
QBASIC	SioLinputT(port%, buf$, term%, len%)
CLIPPER	sio_line_t(port, buf, term)
Assembly	_ _sio_linput_t
Windows	int FAR PASCAL sio_linput_t(port, char far *buf, len, term)

Argument:

int port, len, term; char *buf;

port (dx) = port number
buf (es:di) = receive buffer pointer with required space
len (cx) = buffer length
term (al) = terminator character

Return:

int n (ax)

n => 0: length of data received
n = -1: no such port
n = -2: time out
n = -3: bad function call
n = -5: port is not open

Example: (C language)

```
int stat, port = 1, len = 40, term = 13;
char buf[40];
stat = sio_timeout(18); /* set 1 sec timeout */
stat = sio_linput_t(port, buf, len, term);
```

Function 23 sio_putch()

Write a character into the driver's output buffer.

Language	Command
C	sio_putch(int port, int code)
PASCAL	sio_putch(port, code:integer) :integer
QBASIC	SioPutch(port%, code%, er%)
CLIPPER	sio_putch(port, code)
Assembly	_ _sio_putch
Windows	int FAR PASCAL sio_putch(port, code)

Argument:

int port, code;

port (dx) = port number
code (al) = character (0-255)

Return:

int n (ax)

n = 1: ok
n = 0: buffer full
n = -1: no such port
n = -3: bad function call
n = -5: port is not open

Example: (C language)

```
int stat, port, code = 13;
stat = sio_putch(port, code);
```

Function 24 sio_putb()

Put a block of data in the driver's output buffer. If the output buffer free space is less than the block length, it will return zero and no data is written. See also sio_write().

Language	Command
C	sio_putb(int port, char *buf, int len)
PASCAL	sio_putb(port:integer; buf:string) :integer
QBASIC	SioPutb(port%, buf$, len%)
CLIPPER	sio_putb(port, buf)
Assembly	_ _sio_putb
Windows	int FAR PASCAL sio_putb(port, char far *buf, len)

Argument:

int port, len; char *buf;

port (dx) = port number
buf (ds:si) = transmit string pointer
len (cx) = transmit string length

Return:

int n (ax)

n = len: length of data transmitted
n = 0 : free buffer not enough
n = -1 : no such port
n = -3 : bad function call
n = -5 : port is not open
n = -6 : data block too large, larger than the Tx buffer of the driver, adjust the data block size

Example: (C language)

```
int stat, port, len = 14;
char *buf = "This is a test";
stat = sio_putb(port, buf, len);
```

Function 25 sio_write()

Put a block of data to the driver's output buffer. The actual length of data written depends on the amount of free space in the driver's output buffer. See also sio_putb().

Language	Command
C	sio_write(int port, char *buf, int len)
PASCAL	sio_write(port:integer; buf:string) :integer
QBASIC	SioWrite(port%, buf$, len%)
CLIPPER	sio_write(port, buf)
Assembly	_ _sio_write
Windows	int FAR PASCAL sio_write(port, char for *buf, len)

Argument:

int port, len; char *buf;

port (dx) = port number
buf (ds:si) = transmit string pointer
len (cx) = transmit string length

Return:

int n (ax)

n > 0: length of data transmitted
n = 0: buffer full
n = -1: no such port
n = -3: bad function call
n = -5: port is not open

Example: (C language)

```
int stat, port = 0, len = 14;
char *buf = "This is a test";
stat = sio_write(port, buf, len);
```

Function 26 sio_putb_t()

Write a block of data into the driver's output buffer with a time out check. See also sio_putb().

Language	Command
C	sio_putb_t(int port, char *buf, int len)
PASCAL	sio_putb_t(port:integer; buf:string) :integer
QBASIC	SioPutbT(port%, buf$, er%)
CLIPPER	sio_putb_t(port, buf)
Assembly	_ _sio_putb_t
Windows	int FAR PASCAL sio_putb_t(port, char far *buf, len)

Argument:

int port, len; char *buf;

port (dx) = port number
buf (ds:si) = transmit string pointer
len (cx) = transmit string length

Return:

int n (ax)

n = len: data transmitted
n = -1 : no such port
n = -2 : time out
n = -3 : bad function call
n = -6 : data block too large, than the Tx buffer of the driver. Please adjust the data block size

Example: (C language)

```
int stat, port = 1, len = 40; char buf[40];
stat = sio_timeout(18); /* set 1 sec timeout */
stat = sio_putb_t(port, buf, len);
```

Function 27 (N) sio_putb_x()

Set RTS state on before writing a block of data into the driver's output buffer. After writing the data into the driver's output buffer, wait "tick" tick (55 ms) then turn RTS off. This function is useful for RS-485 comm.

Language	Command
C	sio_putb_x(int port, char *buf, int len, int tick)
PASCAL	sio_putb_x(port:integer; buf:string, len:integer, tick:integer):integer
QBASIC	SioPutbX(port%, buf$, er%)
CLIPPER	sio_putb_x(port, buf)
Assembly	_ _sio_putb_x
Windows	int FAR PASCAL sio_putb_x(port, char far *buf, len, tick)

Argument:

int port, len, tick; char *buf;

port (dx) = port number
buf (ds:si) = transmit string pointer
len (cx) = transmit string length
tick (ax) = delay time (ticks)

Return:

int n (ax)

n = len: data transmitted
n = 0 : free buffer not enough
n = -1 : no such port
n = -3 : bad function call
n = -5 : port is not open
n = -6 : data block large than the driver's output buffer, please adjust the data block size

Example: (C language)

```
int stat, port = 1, len = 40; char buf[40];
stat = sio_putb_x(port, buf, len, 1);
```

Function 28 sio_lstatus()

Get the status of the line.

Language	Command
C	sio_lstatus(int port)
PASCAL	sio_lstatus(port:integer) :integer
QBASIC	SioLstatus(port%, flag%)
CLIPPER	sio_lstatu(port)
Assembly	_ _sio_lstatus
Windows	int FAR PASCAL sio_lstatus(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n => 0: line status:
bit 0ßĎS_CTS
bit 1ßĎS_DSR
bit 2ßĎS_RI
bit 3ßĎS_CD
n = -1: no such port
n = -2: can't control MOXA board
n = -3: bad function call

Example: (C language)

```
#include "head-c.h"
int stat, port = 1;
stat = sio_lstatus (port);
if ((stat & S_DSR) == 2) printf ("DSR ON\n");
else printf ("DSR OFF\n");
```

Function 29 sio_iqueue()

Get the size of data accumulated in the driver's input buffer.

Language	Command
C	sio_iqueue(int port)
PASCAL	sio_iqueue(port:integer) :integer
QBASIC	SioIQueue(port%, size%)
CLIPPER	sio_iqueue(port)
Assembly	_ _sio_iqueue
Windows	long FAR PASCAL sio_iqueue(port)

Argument:

```
long sio_iqueue();
int port;

port (dx) = port number
```

Return:

```
long int n (ax)

n >  0:   data in input buffer (bytes)
n = -1:   no such port
n = -3:   bad function call
n = -5:   port is not open
```

Example: (C language)

```
int port = 1; long stat;
stat = sio_iqueue(port);
if (stat > 0)
      printf ("%ld bytes data in receive buffer\n", stat);
```

Function 30 sio_oqueue()

Get the length of data not yet sent out in the driver's output buffer. (User must be aware of the fact that there may have a couple of characters still kept in RS-232 UART chip and not yet transmitted when the sio_oqueue() return a zero value.)

Language	Command
C	sio_oqueue(int port)
PASCAL	sio_oqueue(port:integer) :integer
QBASIC	SioOQueue(port%, size%)
CLIPPER	sio_oqueue(port)
Assembly	_ _sio_oqueue
Windows	long FAR PASCAL sio_oqueue(port)

Argument:

int port;

port (dx) = port number

Return:

long int n (ax)

n => 0: length of data still kept in the driver's output buffer
n = -1: no such port
n = -3: bad function call
n = -5: port is not open

Example: (C language)

```
int port = 1;
long stat;
stat = sio_oqueue(port);
if (stat > 0)
        printf ("%ld bytes data in transmit buffer\n", stat);
```

Function 31 sio_ifree()

Get the length of free space in driver's input buffer.

Language	Command
C	sio_ifree(int port)
PASCAL	sio_ifree(port:integer) :integer
QBASIC	SioIFree(port%, flag%)
CLIPPER	sio_ifree(port)
Assembly	_ _sio_ifree
Windows	long FAR PASCAL sio_ifree(port)

Argument:

int port;

port (dx) = port number

Return:

long int n (ax)

n => 0: free space in input buffer
n = -1: no such port
n = -3: bad function call
n = -5: port is not opened

Example: (C language)

```
int port = 1;
long stat;
stat = sio_ifree(port);
if (stat > 0)
        printf ("Free receive buffer size=%d\n", stat);
```

Function 32 sio_ofree()

Get the length of free space in the driver's output buffer.

Language	Command
C	sio_ofree(int port)
PASCAL	sio_ofree(port:integer) :integer
QBASIC	SioOFree(port%, flag%)
CLIPPER	sio_ofree(port)
Assembly	_ _sio_ofree
Windows	long FAR PASCAL sio_ofree(port)

Argument:

int port;

port (dx) = port number

Return:

long int n (ax)

n => 0: free space in output buffer
n = -1: no such port
n = -3: bad function call
n = -5: port is not open

Example: (C language)

```
int port = 1; long stat;
stat = sio_ofree(port);
if (stat > 0)
        printf ("Free transmit buffer size = %ld\n", stat)
```

Function 33 sio_Tx_hold()

Check the reason why data could not be transmitted.

Language	Command
C	sio_Tx_hold(int port)
PASCAL	sio_Tx_hold(port:integer) :integer
QBASIC	SioTxHold(port%, flag%)
CLIPPER	sio_Txhold(port)
Assembly	_ _sio_Tx_hold
Windows	int FAR PASCAL sio_Tx_hold(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n : bit 0 onßĎcould not transmit data because CTS is low
bit 1 onßĎcould not transmit data because XOFF char received
bit 2 onßĎcould not transmit data because sio_disableTx() function is called
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
stat = sio_Tx_hold(port);
if (stat == 1)
      printf ("Tx disabled by CTS low\n")
```

Function 34 sio_getbaud()

Get the serial ports baud rate setting. The return value is the actual baud rate. For example, return value 9600 means 9600 bps while 200 means 200 bps.

Language	Command
C	sio_getbaud(int port)
PASCAL	getbaud(port:integer) :integer
QBASIC	SioGetBaud(port%, baud%)
CLIPPER	sio_gbaud(port)
Assembly	_ _sio_getbaud
Windows	int FAR PASCAL sio_getbaud(port)

Argument:

long port;

port (dx) = port number

Return:

int n (ax)

n > 0: the actual baud rate
n = -1: no such port
n = -3: bad function call

Example (C language):

```
int, port = 1; long baud;
baud = sio_getbaud(port);
```

Function 35 sio_getmode()

Get the serial port's mode setting. Please refer to sio_ioctl() for the mode setting.

Language	Command
C	sio_getmode(int port)
PASCAL	sio_getmode(port:integer) :integer
QBASIC	SioGetMode(port%, mode%)
CLIPPER	sio_gmode(port)
Assembly	_ _sio_getmode
Windows	int FAR PASCAL sio_getmode(port)

Argument:

int port

port (dx) = port number

Return:

int n (ax)

n => 0: mode (see sio_ioctl())
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
stat = sio_getmode(port);
```

Function 36 sio_getflow()

Get the serial port's hardware and software flow control setting. See the sio_flowctrl() function.

Language	Command
C	sio_getflow(int port)
PASCAL	sio_getflow(port:integer) :integer
QBASIC	SioGetFlow(port%, flow%)
CLIPPER	sio_gflow(port)
Assembly	_ _sio_getflow
Windows	int FAR PASCAL sio_getflow(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

bit 0= CTS flow control
bit 1= RTS flow control
bit 2= Tx XON/XOFF flow control
bit 3= Rx XON/XOFF flow control
(0 = OFF; 1 = ON)

n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
stat = sio_getflow(port);
switch (stat) {
        case 0:   printf ("Port 1 flow control OFF\n"); break;
        case 1:   printf ("Port 1 CTS flow control ON\n"); break;
        .........
}
```

Function 37 sio_brk_cnt()

Get the counter value of received BREAK signal and then clear.

Language	Command
C	sio_brk_cnt(int port)
PASCAL	sio_brk_cnt(port:integer) :integer
QBASIC	SioBrkCnt(port%, cnt%)
CLIPPER	sio_brkcnt(port)
Assembly	_ _sio_brk_cnt
Windows	int FAR PASCAL sio_brk_cnt(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n => 0: break count
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
stat = sio_brk_cnt(port);
if (stat >= 0)
        printf ("%d counts break signal received\n", stat);
else
        printf ("Fail to access\n");
```

Function 38 sio_overflow()

Check if driver's input buffer is overflow.

Language	Command
C	sio_overflow(int port)
PASCAL	sio_overflow(port:integer) :integer
QBASIC	SioOverflow(port%, er%)
CLIPPER	sio_oflow(port)
Assembly	_ _sio_overflow
Windows	int FAR PASCAL sio_overflow(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n = 0: no overflow
n = 1: received data overflow
n = -1: no such port

Example: (C language)

```
int port = 1;
stat = sio_overflow(port);
if (stat == 1)
        printf ("Port#%d received data overflow\n", port);
```

Function 39 (N) sio_data_status()

Check if any error happened when receiving data.

Language	Command
C	sio_data_status(int port)
PASCAL	sio_data_status(port:integer) :integer
QBASIC	SioDataStatus(port%, er%)
CLIPPER	sio_data_status(port)
Assembly	_ _sio_data_status
Windows	int FAR PASCAL sio_data_status(port)

Argument:

int port;

port (dx) = port number

Return:

int n (ax)

n > 0: bit 0 onßĎparity error
bit 1 onßĎframing error
bit 2 onßĎoverrun error
bit 3 onßĎoverflow error
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int port = 1;
stat = sio_data_status(port);
```

Function 40 sio_term_irq()

Set an interrupt service rountine for the case when terminator character is received. The service routine must read all the data in the input buffer when the interrupt is issued. When "terminator" is encountered, the system will be interrupted and will call the interrupt service routine. Once the interrupt service is complete, the system will return to the point where it was interrupted. (Note that some DOS functions, e.g. printf(), are not allowed to be used in the interrupt service routine.)

Language	Command
C	sio_term_irq(int port, interrupt (*func)(), char code)
PASCAL	----
QBASIC	----
CLIPPER	----
Assembly	_ _sio_term_irq
Windows	int FAR PASCAL sio_term_irq(port, far_p, terminator)

Argument:

int port; char code; interrupt (*func)();

port (dx) = port number
code (al) = terminator code
func (es:di) = service function entry

FARPROC far_p;
int terminator;

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
char code = 13;
void interrupt func (void);
stat = sio_term_irq(port, func, code);
```

Example: (Windows MSC)

```
void FAR PASCAL term_isr();
far_p = MakeProcInstance (term_isr, hInst);
sio_term_irq(port, far_p, 13);
.
.
.
void FAR PASCAL term_isr()
{
        /* interrupt service routine */
}
```

Function 41 sio_cnt_irq()

Set an interrupt service routine for the case when a certain amount of data has been received. When there are `count' bytes of data received in the input buffer, the system will be interrupted and will call the interrupt service routine. Once the interrupt has been served, the system will return to the point where it was interrupted. (Note that some DOS functions, e.g. printf(), are not allowed to be used in the interrupt service routine.)

Language	Command
C	sio_cnt_irq(int port, interrupt (*func)(), int count)
PASCAL	----
QBASIC	----
CLIPPER	----
Assembly	_ _sio_cnt_irq
Windows	int FAR PASCAL sio_cnt_irq(port, far_p, count)

Argument:

int port, count; interrupt (*func)();

port (dx) = port number
count (cx) = data count
func(es:di) = service function entry

FARPROC far_p;

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1, count = 255;
void interrupt func (void);
.
```

```
.
stat = sio_cnt_irq(port, func, count);
.
.
void interrupt func()
{
        /* intrrupt service routine */
}
```

Example: (Windows MSC)

```
void FAR PASCAL cnt_isr();
far_p = MakeProcInstance (cnt_isr, hInst);
sio_cnt_irq(port, far_p, count)
.
.
.
void FAR PASCAL cnt_isr()
{
        /* intrrupt service routine */
}
```

Function 42 sio_modem_irq()

Set an interrupt service routine for the case when the line status is changed. When line status (CTS, DSR, CD, RI) changes, the system will be interrupted and call the interrupt service routine. After the interrupt has been served, the system will return to the point where it was interrupted. (Note that some DOS functions, e.g. printf(), are not allowed to be used in the interrupt service routine.)

Language	Command
C	sio_modem_irq(int port, interrupt (*func)())
PASCAL	----
QBASIC	----
CLIPPER	----
Assembly	_ _sio_modem
Windows	int FAR PASCAL sio_modem_irq(port, far_p)

Argument:

int port, interrupt (*func)();

port (dx) = port number
func (es:di) = service function entry

FARPROC far_p;

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call

Example: (C language)

```
int stat, port = 1;
void interrupt func (void);
.
.
stat = sio_modem_irq(port, func);
```

```
.
.
void interrupt func()
{
        /* intrrupt service routine */
}
```

Example: (Windows MSC)

```
void FAR PASCAL modem_isr();
far_p = MakeProcInsatance (modem_isr, hInst);
sio_modem_irq(port, far_p)
.
.
.
void FAR PASCAL modem_isr()
{
        /* intrrupt service routine */
}
```

Function 43 sio_break_irq()

Set an interrupt service routine for the case when a BREAK signal is received. When a BREAK signal is encountered, the system will be interrupted and will call interrupt service routine. Once the interrupt service is complete, the system will return to the point where it was interrupted. (Note that some DOS functions, e.g. printf(), are not allowed to be used in the interrupt service routine.)

Language	Command
C	sio_break_irq(int port, interrupt (*func)())
PASCAL	----
QBASIC	----
CLIPPER	----
Assembly	_ _sio_break_irq
Windows	int FAR PASCAL sio_break_irq(port, far_p)

Argument:

```
int port;
interrupt (*func)();

port (dx) = port number
func (es:di) = service function entry

FARPROC far_p;
```

Return:

```
int n (ax)

n =  0:   ok
n = -1:   no such port
n = -3:   bad function call
```

Example: (C language)

```
int stat, port = 1;
void interrupt func (void);
.
.
```

```
stat = sio_break_irq(port, func);
.
.
void interrupt func()
{
        /* interrupt service routine */
}
```

Example: (Windows MSC)

```
void FAR PASCAL break_isr();
far_p = MakeProcInstance (break_isr, hInst);
sio_break_irq(port, far_p);
.
.
.
void FAR PASCAL break_isr()
{
        /* intrrupt service routine */
}
```

Function 44 (N) sio_McAnswer()

This function answers the caller manually by sending "ATA" command to the modem and then wait for carrier. After the command is invoked, the application program can use sio_McIsConnect() to see if a "CONNECT" message comes in and take appropriate action.

Language	Command
C	sio_McAnswer(int port)

Argument:

int port;

port = port number

Return:

int n

n > 0: connected
n = -1: no such port or port not opened
n = -2: timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McAnswer(port);
```

Function 45 (N) sio_McAutoAnswer()

Automatically go off hook and answer the caller after a number of rings.

Language	Command
C	sio_McAutoAnswer(int port, int rings)

Argument:

int port, rings;

port = port number
rings = the number of rings before

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1, rings = 3;

stat = sio_McAutoAnswer(port, rings);    /* answer caller after 3 rings */
```

Function 46 (N) sio_McCommandMode()

Tell the modem to go off line. This function tells the modem to go off line by sending "+++" string.

Language	Command
C	sio_McCommandMode(int port)

Argument:

int port;

port = port number

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McCommandMode(port);
```

Function 47 (N) sio_McDataMode()

Tell the modem to go on line. This function tells the modem to go on line by sending "ATO" string.

Language	Command
C	sio_McDataMode(int port)

Argument:

int port;

port = port number

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McDataMode(port);
```

Function 48(N) sio_McDial()

Dial a phone number and wait for carrier. By default, the command will be sent in the format "ATDTnnnnnnn\r", where nnnnnnn is the phone number.

If pulse dialing is required, add a "P" in front of the phone number. The command will be sent in "ATDPnnnnnnn\r" format in this case.

Language	Command
C	sio_McDial(int port, char *phone_number)

Argument:

int port; char *phone_number;

port = port number
phone_number = phone number

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McDial(port, "2134567");
```

Function 49 (N) sio_McGetSRegister()

Read the value of one of the modem S registers. The format of the command will be "ATSnn?\r". This will cause the modem to return a numeric value for the status register, followed by the OK message.

Language	Command
C	sio_McGetSRegister(int port, int sreg)

Argument:

int port, sreg;

port = port number
sreg = the modem S register to read

Return:

int n

n >= 0: the S register's value
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McGetSRegister(port, 7); /* read S register #7 */
```

Function 50 (N) sio_McHangUp()

Hang up the line. This function will hang up the line by sending "ATH0" command.

Language	Command
C	sio_McHangUp(int port)

Argument:

int port;

port = port number

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McHangUp(port);
```

Function 51 (N) sio_McIsConnect()

Check if the line is connected. This routine will check if there is a "CONNECT" message received. If so, the "CONNECT xxxx" message, "CONNECT 19200" for example, will be stored into the user's buffer.

Language	Command
C	sio_McIsConnect(int port, char *return_msg)

Argument:

int port; char *return_msg;

port = port number
return_msg = the buffer to hold the return message

Return:

int n

n > 0: yes, connected
n = 0: not yet
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1; char return_msg[100];

stat = sio_McIsConnect(port, return_msg);
```

Function 52 (N) sio_McOffHook()

Tell the modem to go off hook. This function will send "ATH1" command to the modem.

Language	Command
C	sio_McOffHook(int port)

Argument:

int port;

port = port number

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port=1;

stat = sio_McOffHook(port);
```

Function 53 (N) sio_McReset()

Perform a reset of the modem by sending the "ATZ" command; "ATV1" for deatil result code. This function also sets the escape code (S register #2) to "+" and escape code guard time (S register #12) to 1 second.

Language	Command
C	sio_McReset(int port)

Argument:

int port;

port = port number

Return:

int n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McReset(port);
```

Function 54 (N) sio_McRings()

Read how many rings has come in.

Language	Command
C	sio_McRings(int port)

Argument:

int port;

port = port number

Return:

int n

n >= 0: the number of rings has come in
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McRings(port);
```

Function 55 (N) sio_McSetSRegister()

Send a register set command to the modem.

Language	Command
C	sio_McSetSRegister(int port, int sreg, int value)

Argument:

int port, sreg, value;

port = port number
sreg = the S register
value = the new value to be assigned to the register

Return:

nt n

n = 0: ok
n = -1: no such port or port not opened
n = -2: AT command timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1;

stat = sio_McSetSRegister(port, 0, 3);    /* set S register #0 to 3 */
```

Function 56 (N) sio_WaitConnect()

Check if the line is connected. This routine will check if there is a "CONNECT" message received. If so, the "CONNECT xxxx" message, "CONNECT 19200" for example, will be stored into user's buffer. Otherwise it will wait for timout time.

Language	Command
C	sio_McWaitConnect(int port, char *return_msg, int timeout)

Argument:

int port, timeout; char *return_msg;

port = port number
return_msg = the buffer to hold the return message
timeout = wait time (seconds) before return

Return:

int n

n > 0: ok
n = -1: no such port or port not opened
n = -2: timeout
n = -3: connect fail, no carrier
n = -4: connect fail, no dialtone
n = -5: connect fail, busy
n = -6: connect fail, no answer

Example: (C language)

```
#include "modem-c.h"

int port = 1, timeout = 3; char return_msg[100];

stat = sio_McWaitConnect(port, return_msg, timeout);
```

Function 57 (N) sio_FtASCIIRx()

Receive a file from the remote end using ASCII file capture. This means there is no protocol being used to receive the file.

Language	Command
C	sio_FtASCIIRx(int port, char *file_name, int (*callback)(long ,int ,char far *, long), int key, int seconds);

Argument:

int port, key, seconds; char *file_name;

port = port number
file_name = the name of the file where the download will be stored
key = abort key defined by user
seconds = timeout time (in seconds)
callback = user-supplied message routine which keeps the user updated on the progress of the download

Input:

```
long  rcvlen;      /*  data length currently received */
int   len;         /*  Rx data length in the buffer       */
char far * buffer; /*  Rx data buffer pointer             */
long  flen;        /*  if > 0 -> the length of the file   */
                   /*  being received                     */
```

Return:

```
>= 0               /*  return >= 0 if wish to continue    */
                   /*  receiving                          */
<  0               /*  return < 0 if wish to stop         */
                   /*  receiving (user abort)             */
```

Return:

int n

n > 0: number of files received
n = 0: send ok
n = -1: no such port
n = -2: protocol timeout
n = -3: user key abort
n = -4: function return abort
n = -5: can not open files

Example: (C language)

```
#include "file-c.h"

nt stat, port = 1, seconds = 300, key = 18; char fname[30];
int callback(rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtASCIIRx(port, fname, callback, key, seconds);
```

Function 58 (N) sio_FtASCIITx()

Transmit an ASCII file to the remote end. This means there is no protocol being used to transmit the file.

Language	Command
C	sio_FtASCIITx(int port, char *file_name, int (*callback)(long ,int ,char far *, long), int key);

Argument:

int port, key; char *file_name;

port = port number
file_name = the name of the file that will be sent
key = abort key defined by user
allback = user-supplied message routine which keeps the user updated on the progress of the upload

Input:

```
long  xmitlen;     /*  data length currently sent          */
int   len;         /*  Rx data length in the buffer        */
char far * buffer; /*  Rx data buffer pointer              */
long  flen;        /*  if > 0 -> current xmit file length  */
```

Return:

```
>= 0               /*  continue to send                    */
<  0               /*  stop sending (user abort)           */
```

Return:

int n

n > 0: number of files received
n = 0: send ok
n = -1: no such port
n = -2: protocol timeout
n = -3: user key abort

n = -4: function return abort
n = -5: can not open files

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtASCIITx(port, "MY_FILE", call_func, key);
```

Function 59 (N) sio_FtKermitRx()

Receive files using Kermit protocol.

Language	Command
C	sio_FtKermitRx(int port, char ** ffname, int fno, int (*callback) (long ,int ,char far *, long), int key);

Argument:

int port, key, fno; char ** ffname;

port = port number
ffname = pointer to a array that hold received file names
fno = max. number of files can be received
key = abort key defined by user
callback = user-supplied message routine which keeps the user updated on the progress of the download

Input:

```
long  rcvlen;      /*  data length currently received */
int   len;         /*  Rx data length in the buffer       */
char far * buffer; /*  Rx data buffer pointer             */
long  flen;        /*  if > 0 : the length of the file    */
                   /*  being received                     */
```

Return:

```
>= 0               /*  continue to receive                */
<  0               /*  stop receiving (user abort)        */
```

Return:

int n

n > 0: number of files received
n = -1: no such port
n = -2: protocol timeout

n = -3: user key abort
n = -4: function return abort
n = -5: can not open files
n = -7: Protocol checking error abort

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18, fno = 2; char fname1[80], fname2[80];
char *ffname[2] = {fname1, fname2};
int callback (rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtKermitRx(port, ffname, fno, callback, key);
```

Function 60 (N) sio_FtKermitTx()

Transmit files using Kermit protocol.

Language	Command
C	sio_FtKermitTx(int port, char *file_name, int (*callback) (long ,int ,char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIITx().

Return:

int n

Refer to sio_FtASCIITx().

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtKermitTx(port, "MY_FILE", call_func, key);
```

Function 61 (N) sio_FtXmodem1KCRCRx()

Receive a file using the XMODEM, 1K block size, 16 bit CRC.

Language	Command
C	sio_FtXmodem1KCRCRx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIIRx().

Return:

int n

n = -7: Protocol checking error abort
others: refer to sio_FtASCIIRx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname[30];
int callback (rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtXmodem1KCRCRx(port, fname, callback, key);
```

Function 62 (N) sio_FtXmodem1KCRCTx()

Transmit a file using XMODEM, 1K block size, 16 bit CRC.

Language	Command
C	sio_FtXmodem1KCRCTx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIITx().

Return:

int n

Refer to sio_FtASCIITx().

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtXmodem1KCRCTx(port, "MY_FILE", call_func, key);
```

Function 63 (N) sio_FtXmodemCheckSumRx()

Receive a file using standard XMODEM and additive checksum calculation.

Language	Command
C	sio_FtXmodemCheckSumRx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIIRx().

Return:

int n

n = -7: Protocol checking error abort
others: refer to sio_FtASCIIRx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname[30];
int callback (rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtXmodemCheckSumRx(port, fname, callback, key);
```

Function 64 (N) sio_FtXmodemCheckSumTx()

Transmit a file using standard XMODEM and additive checksum.

Language	Command
C	sio_FtXmodemCheckSumTx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIITx().

Return:

int n

Refer to sio_FtASCIITx().

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtXmodemCheckSumTx(port, "MY_FILE", call_func, key);
```

Function 65 (N) sio_FtXmodemCRCRx()

Receive a file using standard XMODEM, 16 bit CRC.

Language	Command
C	sio_FtXmodemCRCRx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIIRx().

Return:

int n

n = -7: Protocol checking error abort
others: refer to sio_FtASCIIRx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname[30];
int callback (rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtXmodemCRCRx(port, fname, callback, key);
```

Function 66 (N) sio_FtXmodemCRCTx()

Transmit a file using standard XMODEM, 16 bit CRC.

Language	Command
C	sio_FtXmodemCRCTx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIITx().

Return:

int n

Refer to sio_FtASCIITx().

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtXmodemCRCTx(port, "MY_FILE", call_func, key);
```

Function 67 (N) sio_FtYmodemRx()

Receive files using the YMODEM protocol.

Language	Command
C	sio_FtYmodemRx(int port, char ** ffname, int fno, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key, fno; char ** ffname;

Refer to sio_FtKermitRx().

Return:

int n

n = -6: Ymodem CAN signal abort
others: refer to sio_FtKermitRx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18, fno = 2; char fname1[80], fname2[80];
char *ffname[2] = {fname1, fname2};
int callback (rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtYmodemRx(port, ffname, fno, callback, key);
```

Function 68 (N) sio_FtYmodemTx()

Transmit a file using the YMODEM protocol.

Language	Command
C	sio_FtYmodemTx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIITx().

Return:

int n

n = -6: Ymodem CAN signal abort
n = -7: Protocol checking error abort
others: refer to sio_FtASCIITx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtYmodemTx(port, "MY_FILE", call_func, key);
```

Function 69 (N) sio_FtZmodemRx()

Receive files using the ZMODEM protocol.

Language	Command
C	sio_FtZmodemRx(int port, char ** ffname, int fno, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key, fno; char ** ffname;

Refer to sio_FtKermitRx().

Return:

int n

n = -8: Zmodem remote skip this send file
others: refer to sio_FtKermitRx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18, fno = 2; char fname1[80], fname2[80];
char *ffname[2] = {fname1, fname2};
int callback (rcvlen, len, buffer, flen)
{
long rcvlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtZmodemRx(port, ffname, fno, callback, key);
```

Function 70 (N) sio_FtZmodemTx()

Transmit a file using the ZMODEM protocol.

Language	Command
C	sio_FtZmodemTx(int port, char *file_name, int (*callback) (long, int, char far *, long), int key);

Argument:

int port, key; char *file_name;

Refer to sio_FtASCIITx().

Return:

int n

n = -7: Protocol checking error abort
n = -8: Zmodem remote skip this send file
others: refer to sio_FtASCIITx()

Example: (C language)

```
#include "file-c.h"

int stat, port = 1, key = 18; char fname;
int callback (xmitlen, len, buffer, flen)
{
long xmitlen; int len; char far *buffer; long flen;
.
.
.
}

stat = sio_FtZmodemTx(port, "MY_FILE", call_func, key);
```

Function 71 sio_getports()

Get the number of ports and stores "port no." in a specified array. The "port no.", which is set in the setup program, is actually an identification for each port.

Language	Command
C	sio_getports(int *p_no_array, int array_size)
PASCAL	----
QBASIC	----
CLIPPER	----
Assembly	_ _sio_getports
Windows	int FAR PASCAL sio_getports(int far *p_no_array, array_size)

Argument:

int *p_no_array, array_size;
*p_no_array (ds:si) = port number array table
array_size (cx) = array table size (must be large enough to hold all port numbers)

Return:

int n (ax), integer *p_no_array

n => 0: total number of ports
n = -1: no driver found

Example: (C language)

```
int p_no_array[1024]; int stat;
if ((stat = sio_getports(p_no_array, 1024)) <= 0)
        printf ("No COM port Found!\n");
else
        printf ("%d COM port found.\n", stat);
```

Function 72 sio_break()

Send out a break signal.

Language	Command
C	sio_break(int port, int time)
PASCAL	sio_break(port, time:integer) :integer
QBASIC	SioBreak(port%, time%, er%)
CLIPPER	sio_break(port, time)
Assembly	_ _sio_break
Windows	int FAR PASCAL sio_break(port, time)

Argument:

int port, time;

port (dx) = port number
time (ax) = break time in tics (1/18.2 second)

Return:

int n (ax)

n = 0: ok
n = -1: no such port
n = -3: bad function call
n = -4: xmit disable
n = -5: port is not open

Example: (C language)

```
int stat, port = 1, time = 2;
stat = sio_break(port, time);
if (stat == 0)
        printf ("Send 0.11 second break signal from port 1\n");
else
        printf ("Unable to send break signal from port 1\n");
```

Function 73 sio_bank()

Get the memory mapping address segment to which the MOXA board is mapped.

Language	Command
C	sio_bank(int drvname, int board)
PASCAL	sio_bank(drvname:integer; board:integer) :interger
QBASIC	SioBank(drvname%, board%, bank%)
CLIPPER	sio_bank(drvname, board)
Assembly	_ _sio_bank
Windows	int FAR PASCAL sio_bank(drvname, board)

Argument:

int drvname (dx), board (ax);

drvname = 0x320 (for MOXA-C218 and C320)
baord = board number (1 - 4)

Return:

int n (ax)

n = dual-port memory mapped segment address (0xC800, ...)
n =-1 : if selected board not found

Example: (C language)

```
int drvname = 0x320, card = 1;
stat = sio_bank(drvname, board);
```

Function 74 sio_timeout()

Set the sio_linput_t() and sio_putb_t() time-out time in ticks.

Language	Command
C	sio_timeout(time_tic)
PASCAL	sio_timeout(time_tic:integer) :integer
QBASIC	SioTimeout(time_tic%)
CLIPPER	sio_timout(time_tic)
Assembly	_ _sio_timeout
Windows	int FAR PASCAL sio_timeout(time_tic)

Argument:

int time_tic;

time_tic (ax) = time out tic counts (1 tic=1/18.2 sec.)

Return:

None

Example: (C language)

```
int stat, tic = 2;
stat = sio_timeout(tic);
```

Function 75 sio_loopback()

Serial port internal loopback transmit/receive test. The port will be closed automatically after the loopback function is completed. The port must be re-opened before further operation.

Language	Command
C	sio_loopback(int port, char *buf, int len)
PASCAL	sio_loopback(port:integer; buf:string) :integer
QBASIC	SioLoopBack(port%, buf$, er%)
CLIPPER	sio_loopbk(port, buf)
Assembly	_ _sio_loopback
Windows	int FAR PASCAL sio_loopback(port, char far *buf, len)

Argument:

```
int port, len; char *buf;

port (dx) = port number
buf (ds:si) = test character string buffer pointer
len (cx) = buffer length
```

Return:

```
int n

n (ax) =0:    ok
n >  0:    error
n = -1:    no such port
n = -3:    bad function call
```

Example: (C language)

```
char *buf = "This is test string";
int stat, port = 0;

if (sio_loopback(port, buf, 19) == 0)
        printf ("Port #%d Loopback test O.K.\n", port);
else printf ("Port #%d sio_loopback fail\n", port);
```

Chapter 4 Troubleshooting

This chapter contains information to help you to troubleshoot problems encountered in board setup, DataScope operation and programming.

4.1 Setup Problem Messages

1. "MOXA-XXXX Not Found"

   Possible reasons:

   (a) No such board in the system or board is not properly plugged.
   (b) I/O or memory base address hardware setting conflict with SETUP.EXE.
   (c) I/O or memory base address conflict with other interface card.
   (d) Memory base address conflicts with BIOS shadow RAM. If so, disable shadow RAM or adjust memory base address.
   (e) Memory base address conflicts with certain software driver such as EMM386, QEMM386 or 386MAX memory management. If so, reserve enough space for MOXA boards, e.g. use "X=" option in EMM386 command line.

2. "DEVICE DRIVER CONFIGURATION DATA FILE ERROR!!"

   Reason: Format error in XXXX.CNF file or damage to the XXXX.CNF file. Use the correct version of SETUP.EXE to produce a XXXX.CNF file with the right format.

3. "CAN NOT FIND THE DRIVER CONFIGURATION DATA FILE!!"

   Reason: XXXX.CNF file not found. The device driver and the configuration files must be in the same directory.

4. "SETUP NONE PORT"

   Reason: Each port of the desired driver is set as "None". Run the setup program to enable at least one port.

5. "THIS DEVICE DRIVER ALREADY EXISTS!!"

   Reason: The DOS TSR driver that you are installing is already installed in the system.

6. "MOXA-XXXX is not the last executed driver, it can not be released."

   Reason: Drivers must be released in the reverse order that they were installed. For example, if you execute two drivers in this order:

   SER-DRV.EXE
   MX-DRV.EXE

   The drivers must be released in this order:

   MX-DRV.EXE/Q
   SER-DRV.EXE/Q

4.2 DataScope Problems

1. When using the Data Scope system to monitor communications between two target systems, no data is shown on the monitor screen.

   Possible reasons:
   (a) Cable hookups to the target systems may not be correct.

   (b) The port numbers or flow control options may not be set correctly.

   (c) A trigger condition may have been set, but the condition has not yet occurred.

2. In terminal emulation mode, the screen update speed is quite slow.

   Reason: This is normal. DataScope operates in graphics mode, not text mode, so screen updating may be slow, depending on the speed of your system.

4.3 Programming Problems

1. MOXA board cannot transfer data through its port.

   Reasons:
   (a) Data cannot be transferred through a port unless the port is first opened using sio_open().

   (b) Configuration may be incorrectly set. Run SETUP.EXE, then press F2 to confirm the correct configuration.

2. After using the sio_loopback() function, ports can no longer access data.

   Reason: The sio_loopback() function automatically closes ports after use. You must re-open the ports to access data again.

3. Data was lost during transmission

   Solution: Apply hardware flow control to prevent data loss. The RS-232 cable should have full handshaking signal lines such as RTS, CTS, DTR, DSR, TxD, RxD and GND. Hardware handshaking has to be specified using the setup program, or else the sio_flowctrl() function must be used to tell the driver to use hardware flow control.

   If hardware flow control cannot be applied, software flow control (XON/XOFF) is a solution. However, software flow control should only be used with text data instead of binary data transmission.

4. After receiving a certain amount of data, a receive-bound, interrupt-driven application cannot be interrupted by the MOXA driver, and thus cannot recieve any further data.

   Reason: Each time a MOXA driver issues an interrupt, it may consist of several interrupt conditions. The solution is to read out or serves the data in the driver's input buffer until it is empty within the interrupt service routine.